Stream Cipher

메시지를 암호화하는 Key의 bit stream처럼 흘러나온다.

Jong Hwan Park

Bit-oriented Cipher

- Character-oriented cipher
  - Classical symmetric-key cipher
  - Information is only in text type

- Bit-oriented cipher
  - Modern symmetric-key cipher
  - Information consists of various types of texts, numbers, graphics, audio, and video data
  - Need *encoding* method to convert those data into bits (01100…)
  - ASCII code:
    - m → 01101101
    - d → 01100100
    - n → 01101110

XOR Property (1)

- XOR operation in computer

  0 XOR 0 = 0

  0 XOR 1 = 1 ⇒ (mod 2)

  1 XOR 0 = 1

  1 XOR 1 = 0

- 0 1 0 0 1 1 0 0 … A

  ⊕ 1 0 1 0 1 0 1 0 … B

  ---

  1 1 1 0 0 1 1 0 … C = A ⊕ B

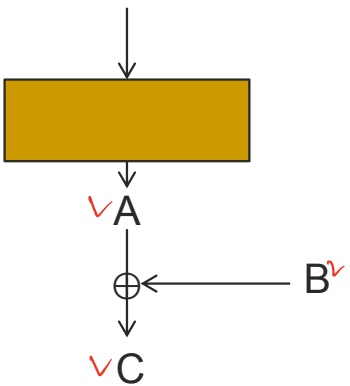
XOR Property (2)

- Cancelation (or masking) effect of XOR

```
  1 1 1 0 0 1 1 0        A ⊕ B = C
⊕ 1 0 1 0 1 0 1 0        B
-------------------------
  0 1 0 0 1 1 0 0        C ⊕ B = A ⊕ B ⊕ B = A
```

message, key

- XOR operation in algorithm



$A \oplus B = C$

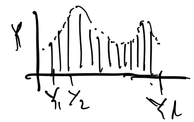
$A \oplus C = B$

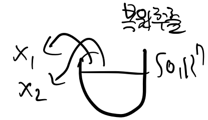
$B \oplus C = A$

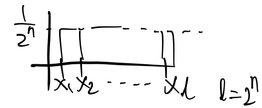
XOR Property (3)

$X \in \{0,1\}^n \Rightarrow X = \overbrace{1010 \cdots 010}^{\#n}$

Thm: Y: a variable over $\{0,1\}^n$, X: an independently uniformly random variable on $\{0,1\}^n$

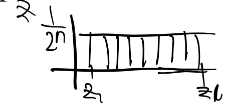






Then $Z := Y \oplus X$ is uniform variable on $\{0,1\}^n$

임의의 분포 (Y) 랜덤분포 (X)



⇒ 평문에 대한 분포 Y는 다 사라지고 정보(의미)가 없는 uniform한 분포로 바뀐다. ⇒ Y의 정보를 숨긴다.

Proof: (for n=1)

Y	Pr
0	P_0
1	P_1

X	Pr
0	1/2
1	1/2

$\Pr[Z=0] = \Pr[(x,y)=(0,0) \text{ or } (x,y)=(1,1)] =$

$\Pr[Z=0 \cap Y=0]$ — 두 사건은 배반사건

$= \Pr[(x,y)=(0,0)] + \Pr[(x,y)=(1,1)] =$

$= \frac{P_0}{2} + \frac{P_1}{2} = \frac{1}{2}$

$\Pr[Z=1] = \Pr[(x,y)=(1,0) \text{ or } (x,y)=(0,1)] =$

$= \Pr[(x,y)=(1,0)] + \Pr[(x,y)=(0,1)]$

$= \frac{P_0}{2} + \frac{P_1}{2} = \frac{1}{2} \quad (\because P_0 + P_1 = 1)$

X	Y	Pr
0	0	$P_0/2$
0	1	$P_1/2$
1	0	$P_0/2$
1	1	$P_1/2$

⇐ X가 random이므로

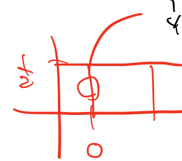
∴ X를 선택할 확률에 따라 Z 결정.

$\Pr[Z=0 \mid Y=0] = \frac{\Pr[Z=0 \cap Y=0]}{\Pr[Y=0]} = \frac{P_0/2}{P_0} = \frac{1}{2}$

$\Pr[Z=0 \mid Y=1] = \frac{\Pr[Z=0 \cap Y=1]}{\Pr[Y=1]} = \frac{P_1/2}{P_1} = \frac{1}{2}$

⇒ Z=0으로 올라가는데 평문이 0에서 1로 왔는지 평문 1에서 0으로 왔는지 알 수 없다. 평문이 균등하게 확률분포.

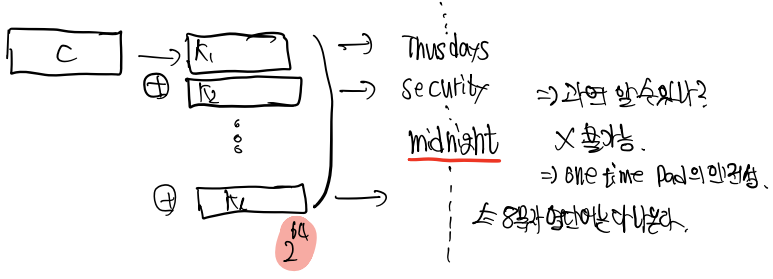
Y가 0일 때 1인지 Y가 1일 때 0인지 확률이 같다.



(Z) 어느 쪽에서 왔는지 모르므로 평문을 알기 어렵다.

One-Time Pad (Vernam's Cipher) (1)

- Vernam presented the one-time pad in 1917
  - *MS*, *KS*, *CS* are all $\{0, 1\}^n$
  - Gen (1^n) → a *random* string k in KS *(with uniform distribution)*
  - Enc (k, m) → c = k ⊕ m
  - Dec (k, c) → m = k ⊕ c

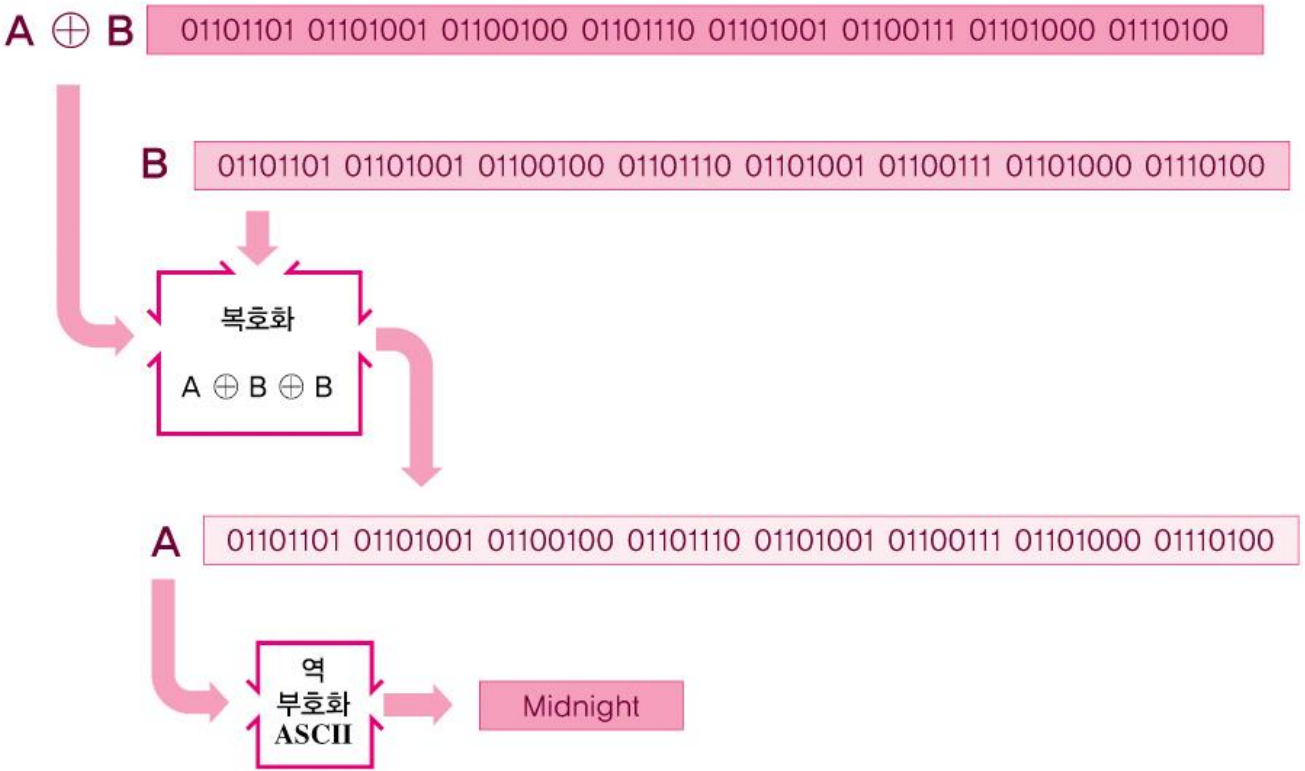


- Encryption of m=midnight

	m	i	d	n	i	g	h	t
ASCII	01101101	01101001	01100100	01101110	01101001	01100111	01101000	01110100
m	midnight							
Key	01101011	11111010	01001000	11011000	01100101	11010101	10101111	00011100
C	00000110	10010011	00101100	10110110	00001100	10110010	11000111	01101000

One-Time Pad (Vernam's Cipher) (2)

- Decryption of the ciphertext



One-Time Pad (Vernam's Cipher) (3)

- One-time pad is *perfectly secure*
  - An adversary who intercepts a ciphertext learns absolutely nothing about the plaintext (→ against ciphertext-only attack)
  - Even an adversary with unbounded computational power cannot learn any information about the plaintext
  - Security does not rely on any of unproven assumptions
  - OTP is the only perfectly secure encryption in cryptography

- Why?
  - For a randomly chosen key with uniform distribution, we have

perfectly secure

$$\Pr[C = c \mid m = m_0] = \frac{1}{2^n} = \Pr[C = c \mid m = m_1]$$

One-Time Pad (Vernam's Cipher) (4)

- Drawbacks of one-time pad
  - The key is required to be as long as the message (|k|=|m|=|c|)
  - Only secure if used once with the same key
    - Why?

      two time pad

      Using the same key k, $c_1 = m_1 \oplus k$ and $c_2 = m_2 \oplus k$

      Then, an adversary has that $c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$

      Enough redundancy in English and ASCII encoding that:

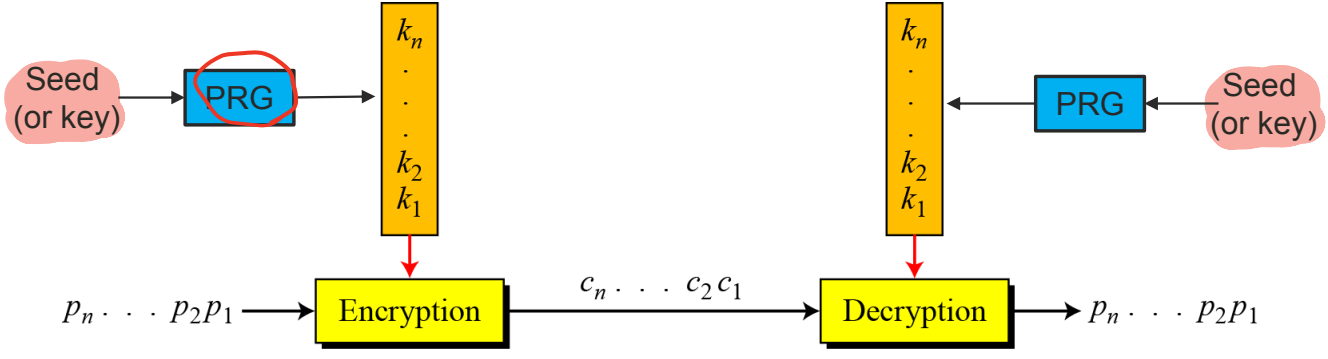
      $m_1 \oplus m_2 \rightarrow m_1 , m_2$

      m_1과 m_2의 내용성X. ⇒ 활용하기 쉽다.
      ⇒ two time pad 활용 (메세지 암호화를 여러번함)
  - Key distribution & key sync.

- Never can be used ?
  - Project Venona 구현의 한계
  - Hot line/

Stream Cipher

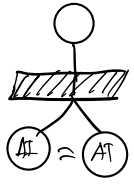
- Idea of stream cipher
  - Inspired by one-time pad

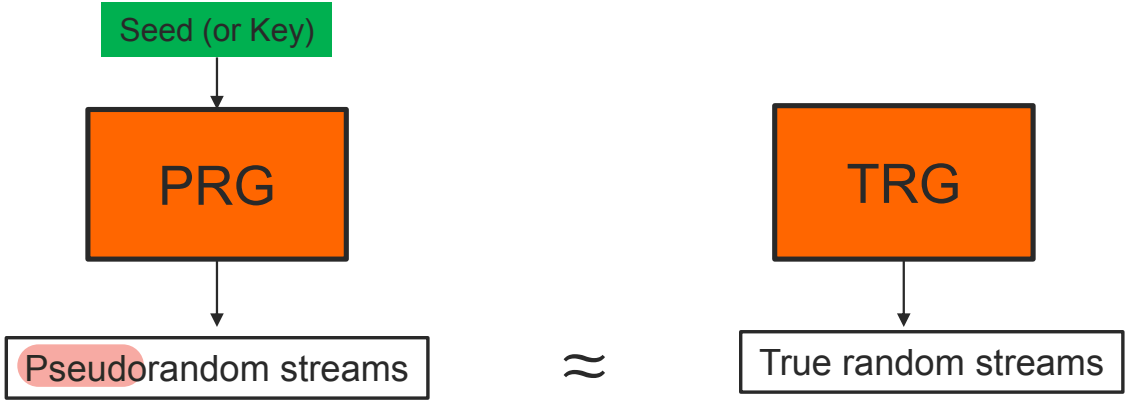


  - Key stream is generated by using a pseudorandom generator (PRG)
  - PRG(short key=seed) → long key stream
  - Encryption is simply XORed with PRG(k) ⊕ m (very fast)
  - Need the short key shared in advance
  - Security depends on how good key streams look 'random'

Security of PRGs

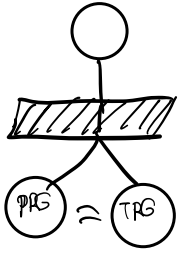
- Indistinguishability between PRG and TRG streams
  - PRG – Pseudo-Random Generator
  - TRG – True Random Generator





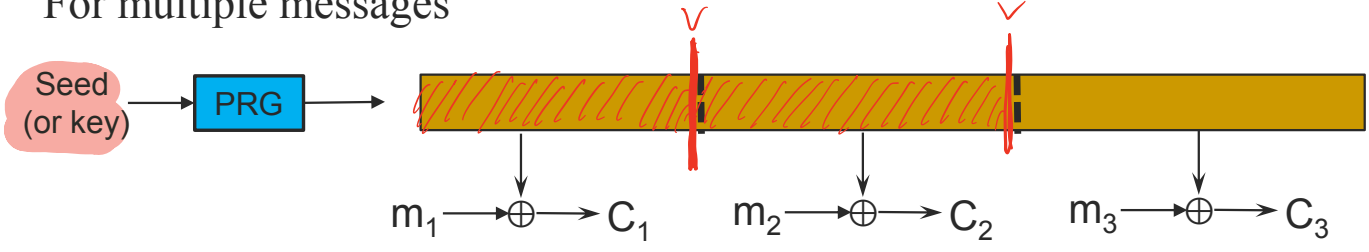
- Randomness check tests for PRG

안전성.



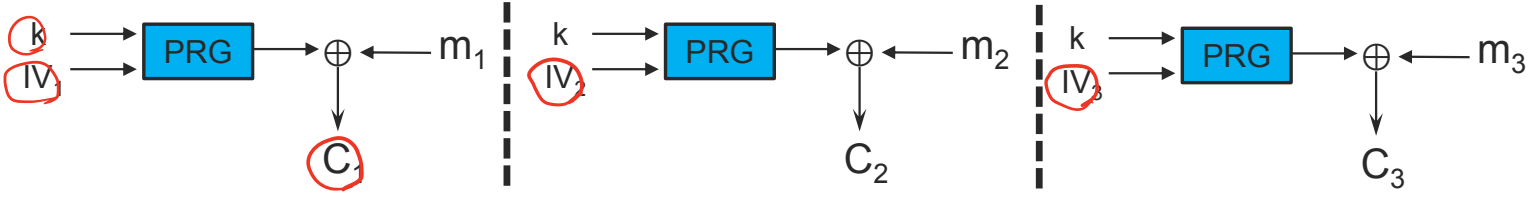
Stream Cipher Modes of Operation

- Synchronized mode
  - Encryption is done by C = PRG(k) ⊕ m
  - For multiple messages



  - Need to maintain synchronized state

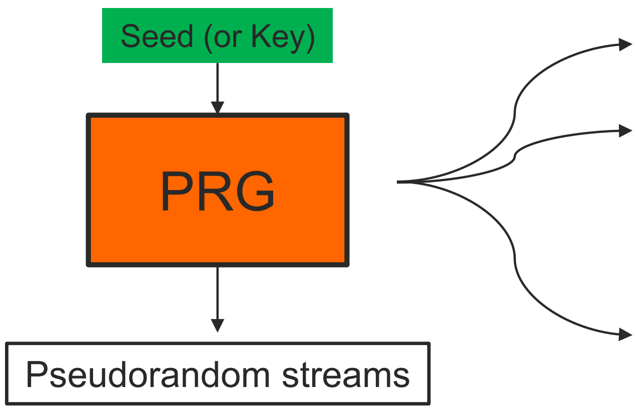
- Unsynchronized mode (stateless way)
  - Encryption is done by C = PRG(k, IV) ⊕ m for initialization vector
  - For multiple messages



Several Approaches to PRGs

PRG들을 안전하게 만드는 방법.

3가지



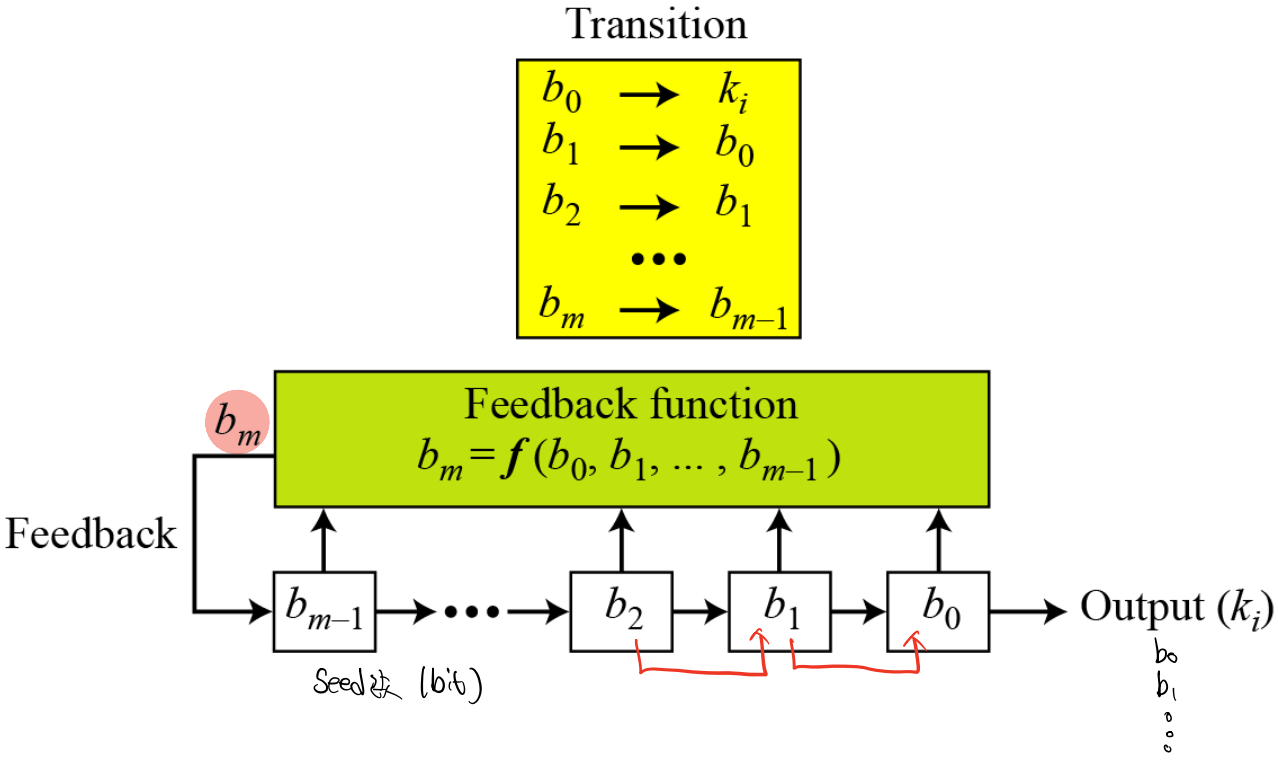
- LFSR-based
- Dedicated to stream cipher
  - RC4, and Salsa, Trivium, and others (from sStream project in Europe)
- Block cipher-based(later)
  - CTR, OFB mode

mode 변경

block cipher 더 분석되고 안전한 block cipher를 가지고 더 성능이 빠른 stream cipher를 만들 수 있다

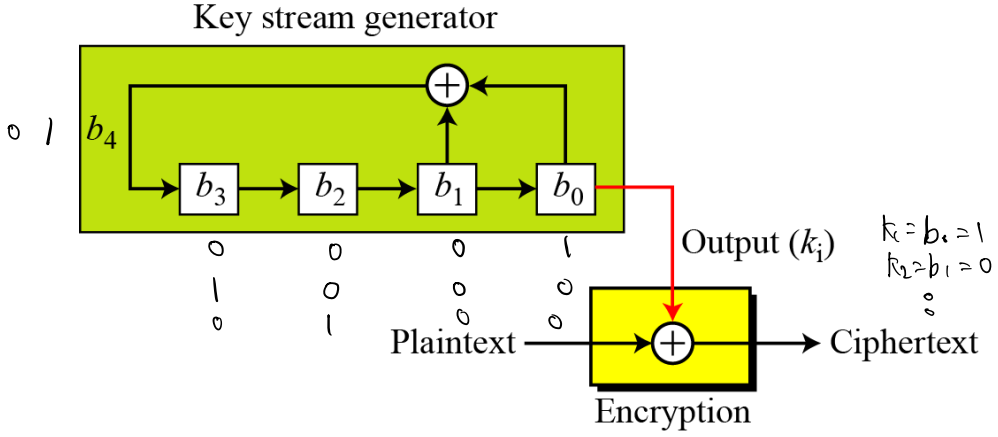
Feedback Shift Register

- Feedback Shift Register (FSR)
  - Seed = (b_{m-1}, …, b_1, b_0) as a secret key



LFSR for PRG

- Linear Feedback Shift Register (LFSR)
  - Seed = $(b_{m-1}, \ldots, b_1, b_0)$
  - $b_m = c_{m-1}b_{m-1} \oplus \cdots \oplus c_2b_2 \oplus c_1b_1 \oplus c_0b_0$ feedback function.
  - Example of LFSR where $b_4 = b_1 \oplus b_0$ if the seed = $(0001)_2$



  - Show the value of output for 20 transitions (shifts)

Outputs of LFSR (1)

States	b_4	b_3	b_2	b_1	b_0	k_i
Initial	1	0	0	0	1	
1	0	1	0	0	0	1
2	0	0	1	0	0	0
3	1	0	0	1	0	0
4	1	1	0	0	1	0
5	0	1	1	0	0	1
6	1	0	1	1	0	0
7	0	1	0	1	1	0
8	1	0	1	0	1	1
9	1	1	0	1	0	1
10	1	1	1	0	1	0

Seed값.

bit열.

Outputs of LFSR (2)

(Continued)

11	1	1	1	1	0	1
12	0	1	1	1	1	0
13	0	0	1	1	1	1
14	0	0	0	1	1	1
15	1	0	0	0	1	1
16	0	1	0	0	0	1
17	0	0	1	0	0	0
18	1	0	0	1	0	0
19	1	1	0	0	1	0
20	1	1	1	0	0	1

(0 0 0 0)

Outputs of LFSR (3)

- The key stream is

100010011010111 **100010011010111** 100010011010111 **100010011010111** …

  - Which looks like a random sequence
  - But it is a repetition of 15 bits

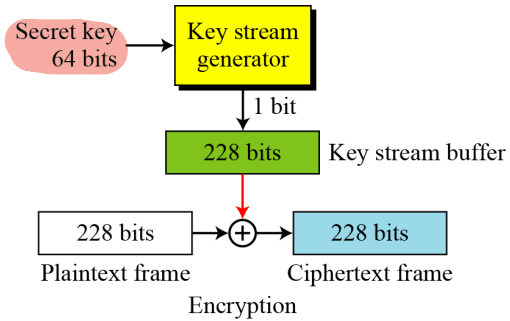
- The key stream from a LFSR with m-bit seed is repeated after N bits
  - Key stream less than N bits acts as a pseudorandom sequence
  - Maximum period $N = 2^m - 1$ (why?)

Seed 자리에 동일 bit stream이 위치하는 곳에 그다음은 동일한 bit stream이 나올 것이다.
= Seed 자리에 동일한 것이 위치하니까

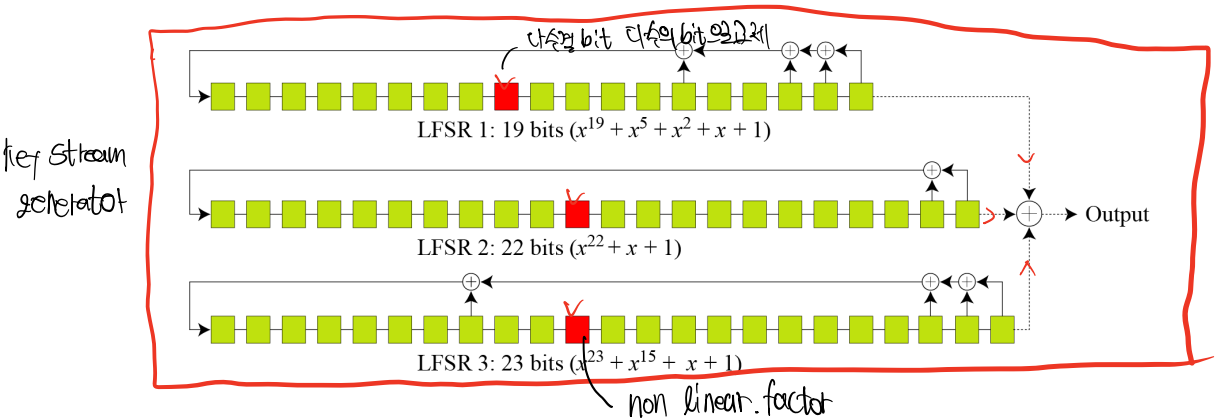
2^m 개 중 중복하여 같은 것이 나온다.

A5/1 using LFSRs

- Used in Global System for Mobile (GSM) communication
  - Security attacks were reported in 2000 and 2003
  - General outline:

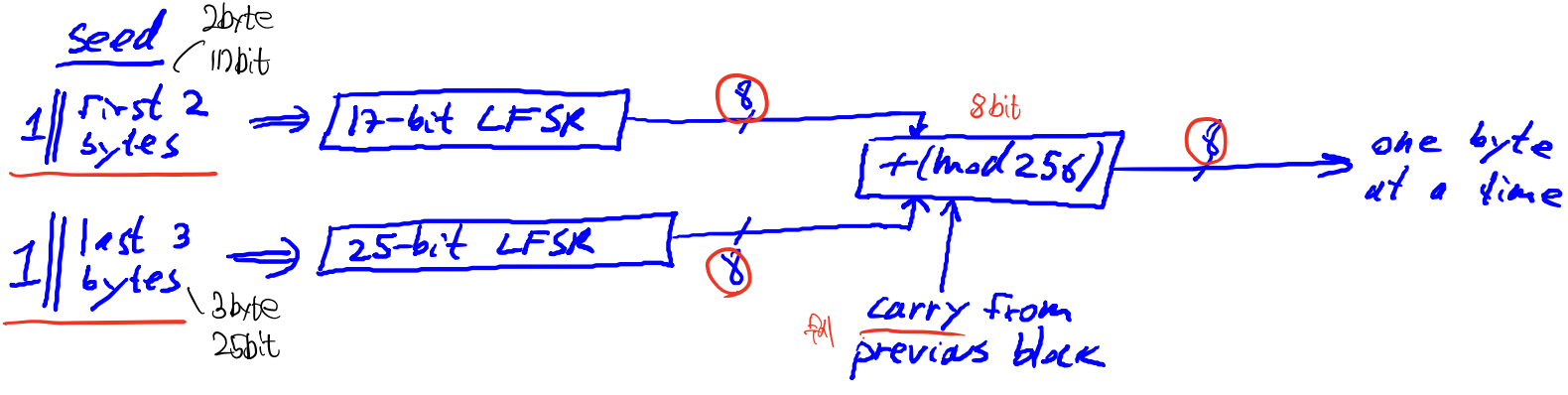


  - Use 3 LFSRs with 19, 22, and 23 bits



Old Example(hardware): CSS

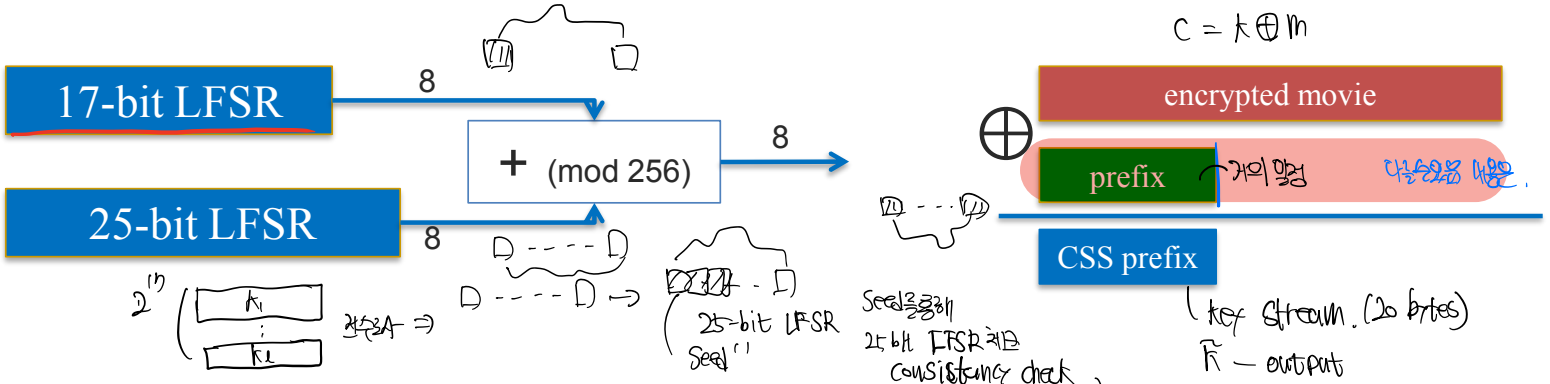
- Used in DVD encryption: badly broken
  - http://en.wikipedia.org/wiki/Content_Scramble_System
- CSS (Content Scrambling System): seed = 5 bytes = 40 bits



Easy to break in time ≃ 2^{17}

Cryptanalysis of CSS

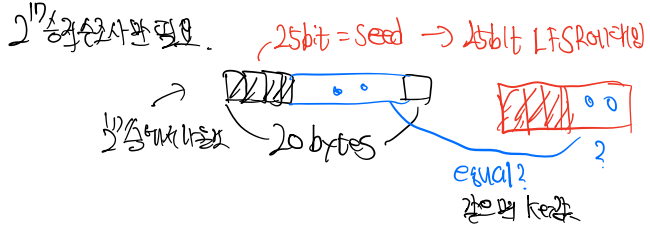
- 2^{17} time attack (using 2^{40}-bit key)



For all possible initial settings of 17-bit LFSR do:

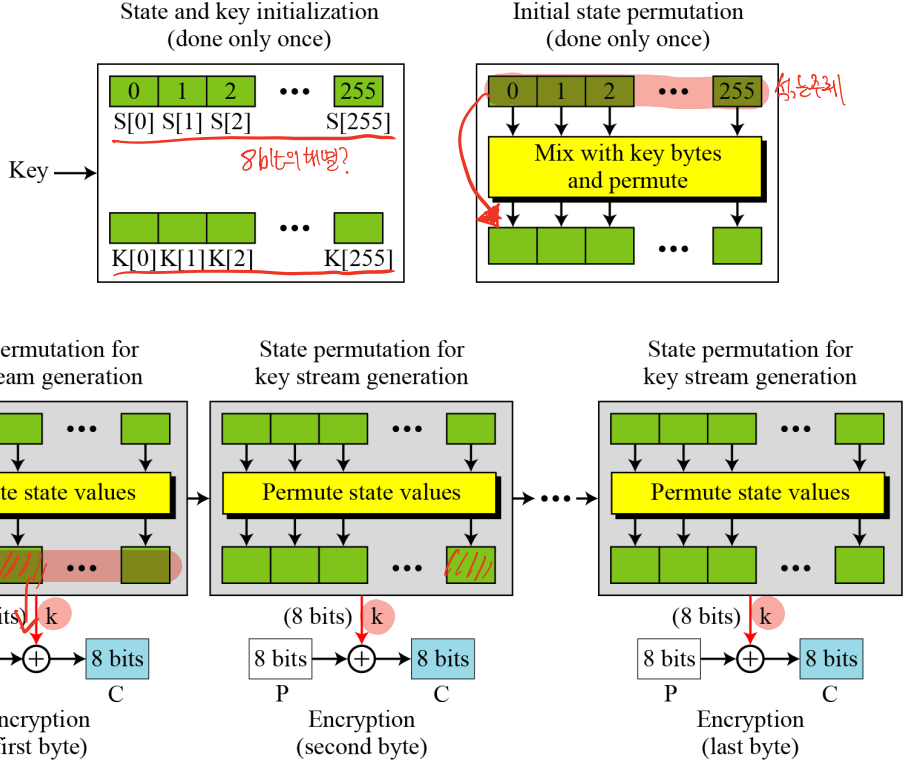
- Run 17-bit LFSR to get 20 bytes of output
- Subtract from CSS prefix ⇒ candidate 20 bytes output of 25-bit LFSR
- If consistent with 25-bit LFSR, found correct initial settings of both !!

Using key, generate entire CSS output



RC4(software)

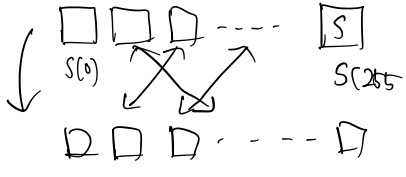
- RC4
  - Designed in 1987 by Ronald Rivest for RSA Data Security
  - Used in SSL/TLS and WEP encryption protocol in IEEE802.11 wireless networks
  - Idea:



RC4 – Encryption Algorithm (1)

Algorithm 8.6 *Encryption algorithm for RC4*

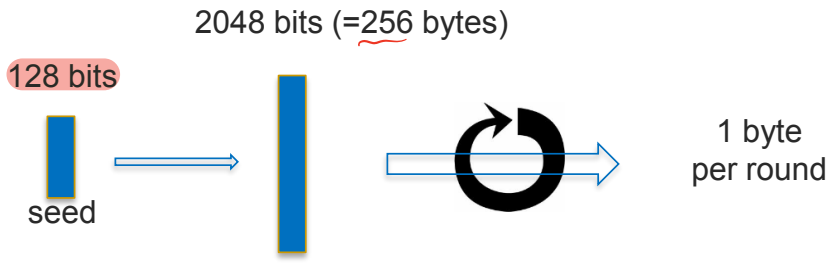
```
RC4_Encryption (K)
{
    // Creation of initial state and key bytes
    for (i = 0 to 255)
    {
        S[i] ← i
        K[i] ← Key [i mod KeyLength]
    }
    // Permuting state bytes based on values of key bytes
    j ← 0
    for (i = 0 to 255)
    {
        j ← (j + S[i] + K[i]) mod 256
        swap (S[i] , S[j])
    }
```



RC4 – Encryption Algorithm (2)

```
    // Continuously permuting state bytes, generating keys, and encrypting
    i ← 0
    j ← 0
    while (more byte to encrypt)
    {
        i ← (i + 1) mod 256
        j ← (j + S[i]) mod 256
        swap (S [i] , S[j])
        k ← S [(S[i] + S[j]) mod 256]
        // Key is ready, encrypt
        input P
        C ← P ⊕ k
        output C
    }
}
```

Cryptanalysis of RC4



- RC4(K=128-bit seed) → randomly-looking key stream
- Weaknesses:
  1. Bias in initial output: Pr[2nd byte = 0] = 2/256
  2. Prob. of (0,0) is $1/256^2 + 1/256^3$
  3. Related key attacks

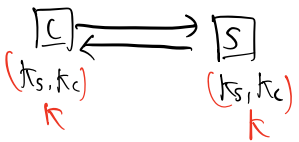
Real Example of Stream Cipher (1)

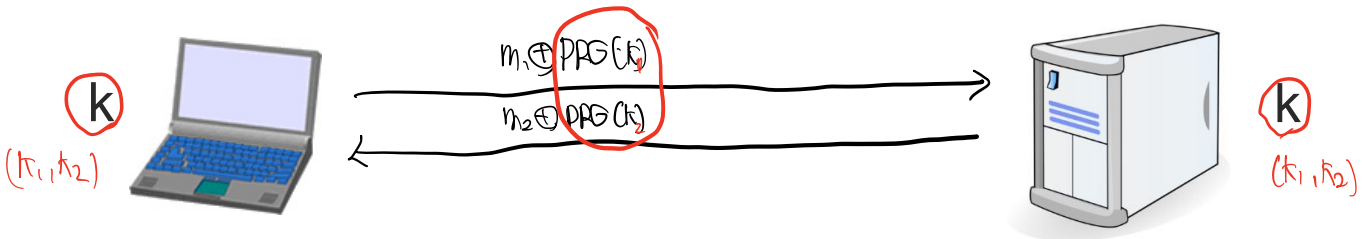
- As in OTP, never use stream cipher key more than once!!

$$C_1 \leftarrow m_1 \oplus PRG(k)$$
$$C_2 \leftarrow m_2 \oplus PRG(k)$$

Eavesdropper does: $C_1 \oplus C_2 \rightarrow m_1 \oplus m_2 \rightarrow m_1, m_2$

- MS-PPTP (windows NT):





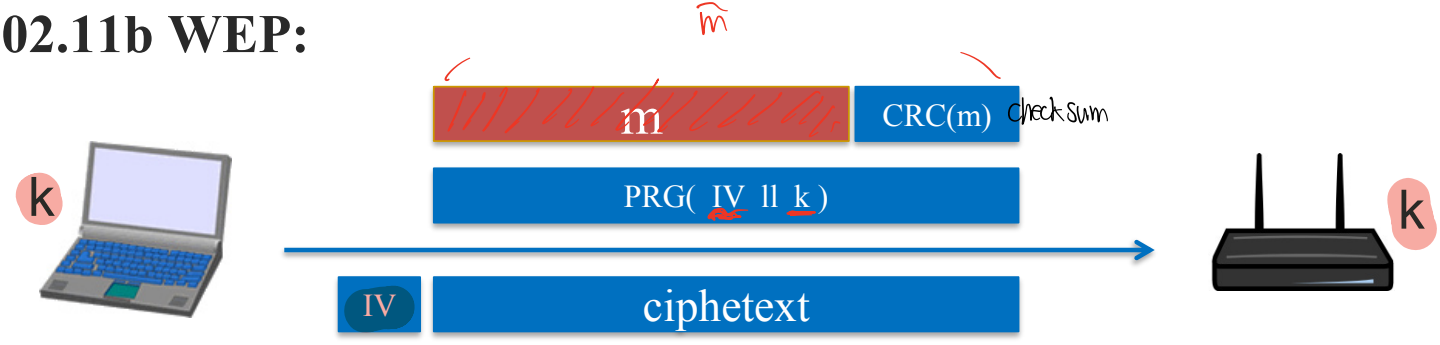
Need different keys for C→S and S→C

k_1 k_2

Real Example of Stream Cipher (2)

PRG ⇒ RC4

802.11b WEP:



자료 WAP
WAP2 사용.

Length of IV: 24 bits

- Repeated IV after $2^{24} \approx 16M$ frames
- On some 802.11 cards: IV resets to 0 after power cycle
  - Key for frame #1: (1||K)
  - Key for frame #2: (2||K)

2^{24} 송수신시 동일한 IV 사용 ⇒ two time pad
RC4의 취약점때문에 해킹할수 있다.

eStream Project

- To identify new stream cipher (efficient and compact) suitable for widespread adoption
- Organized by the EU ECRTPT network
- Running from 2004 to 2008 and updated in 2009 and 2012
- Dedicated for unsynchronized mode
- eStream finalist

PRG(K, IV)

- HC-128
- Rabbit
- Sosemanuk
- Salsa20 현재사용

SW-oriented

- MICKEY-128 v2
- Trivium 현재사용 → LFSR-based
- Grain v1

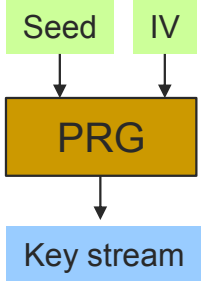
HW-oriented

Modern Stream Ciphers: eStream

- Unsynchronized mode (stateless way)

PRG: $\{0,1\}^s \times R \longrightarrow \{0,1\}^n$

seed, IV = nonce



Nonce: a non-repeating value for a given key

(number used at once) 한번쓰는 random = 한번쓰는 숫자. random이 아니어도괜찮, 한번쓰는 숫자를 말함.

$E(k, m ; r) = m \oplus PRG(k ; r)$ (r: nonce)

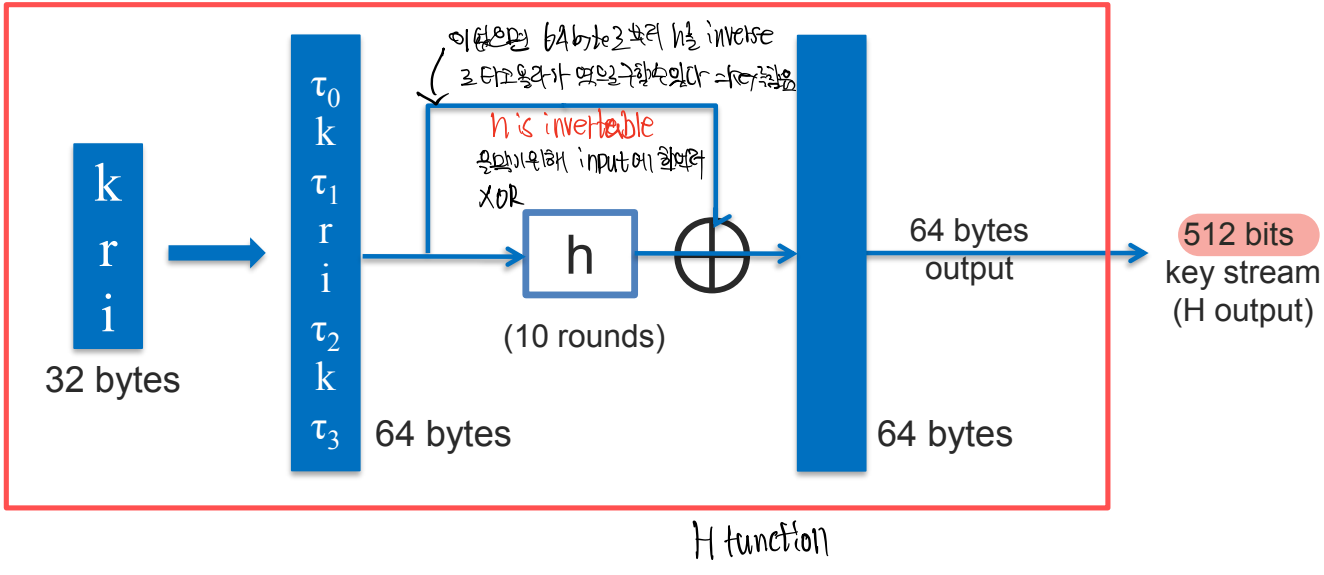
- random #
  → r_1, r_2, r_3
  확률적 = uniform
- nonce
  → 0, 1, 2, 3, 4 ---
  정해진규칙하에 1번만사용.

- The pair (k,r) is never used more than once
- What is the advantage of using nonce? 정해진규칙하에서 두사람이 따로따로시작하여 암호문을 정확하게 정하면은 굳이 네트워크에 돌리지않아도상관없음.

eStream: Salsa20 (SW or HW)

Salsa20: $\{0,1\}^{128 \text{ or } 256} \times \{0,1\}^{64} \longrightarrow \{0,1\}^{n}$ (max $n = 2^{73}$ bits)

Salsa20(k ; r) := H(k , (r, 0)) ll H(k , (r, 1)) ll …



h: invertible function. designed to be fast on x86

Performance

AMD Opteron, 2.2 GHz (Linux)

	PRG	Speed (MB/sec)
	RC4	126
eStream	Salsa20/12	643
eStream	Sosemanuk	727

Q & A